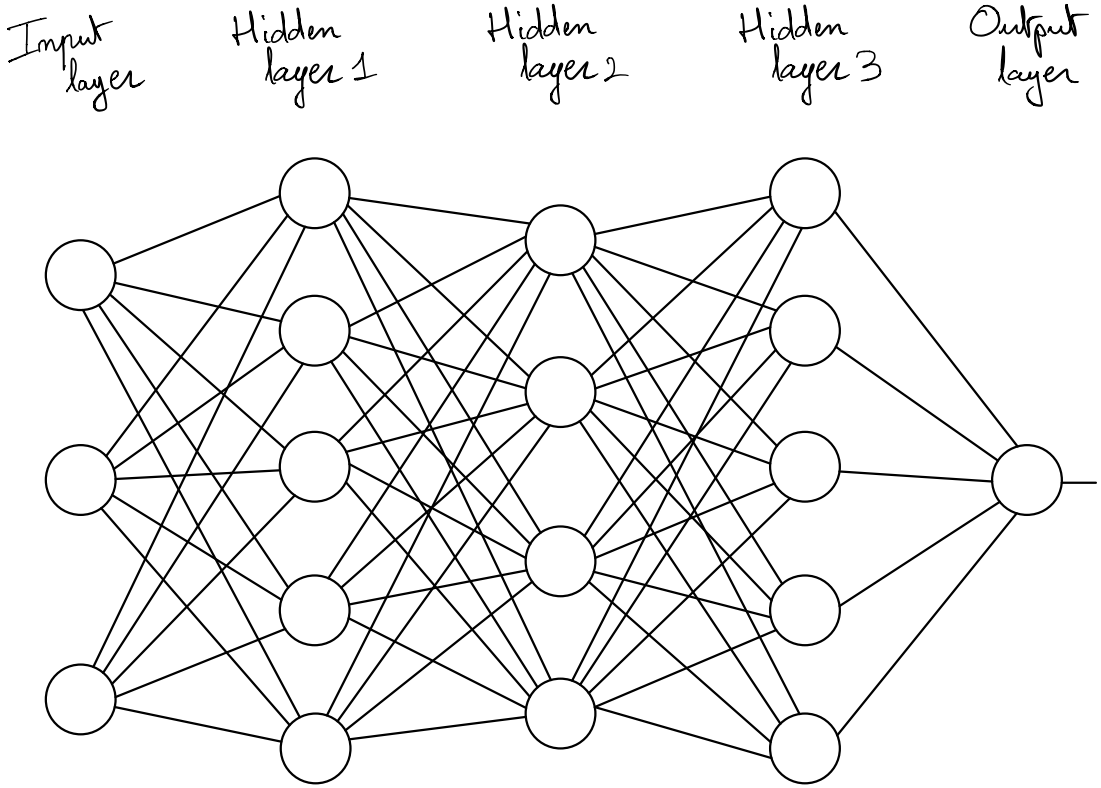
Outline

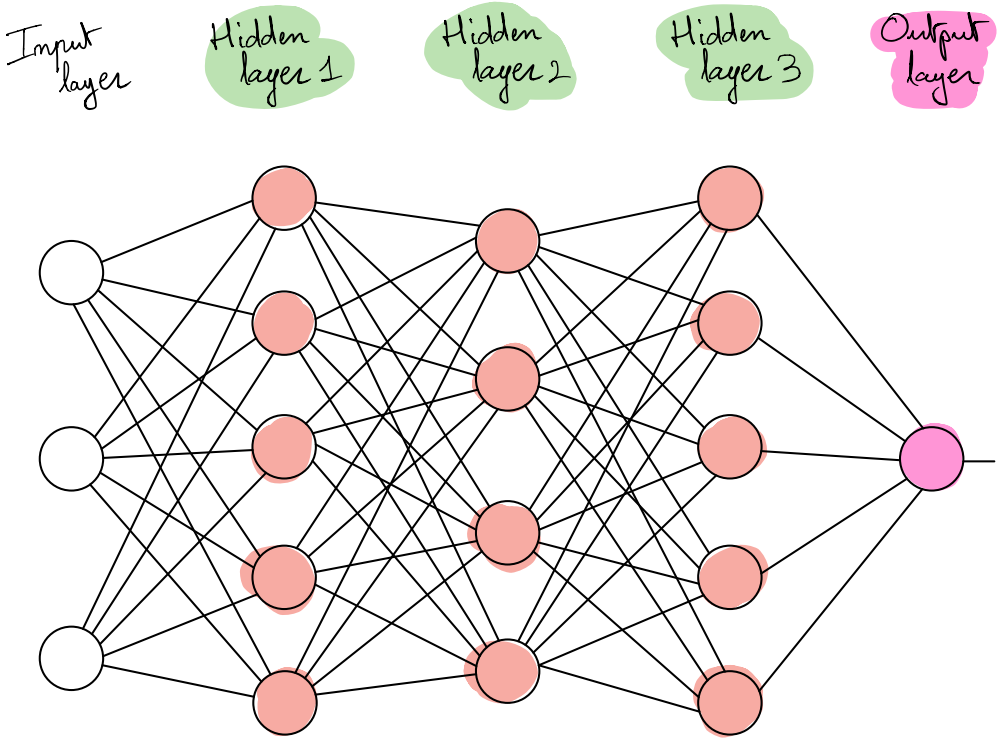
What to set in a neural network?



Network structure:
- Number of layers / neurons per layer
- Activation functions
- Output unit
- Specific layers (dropout, batch normalization)

Optimization:
- Optimization algorithm
- Weights / biases initialization
- Loss function

What to set in a neural network?



Network structure:

- Number of layers / neurons per layer
- Activation functions
- Output unit
- Specific layers (dropout, batch normalization)

Optimization:

- Loss function
- Optimization algorithm
- Weights / biases initialization

Outline

Number of hidden layers/neurons

- No particular rules for choosing the number of layers or the number of neurons per layer.
- Read research papers related to the task you want to solve and test the architecture they propose.
- You may want to change the architecture a bit to see how it influences the performance.
- Beware: there exist many rules of thumbs which are not supported by evidence (either practical or theoretical).

Number of hidden layers/neurons

- Use data-driven strategies:
  - Network pruning following the procedure training/pruning/training/pruning/...
    ["What is the state of neural network pruning?", Blalock et al. 2020]
  - More complex evolutionary algorithms
    ["AgEBO-Tabular: Joint Neural Architecture and Hyperparameter Search with Autotuned Data-Parallel Training for Tabular Data", Egele et al. 2020]

Outline

Sigmoid activation function

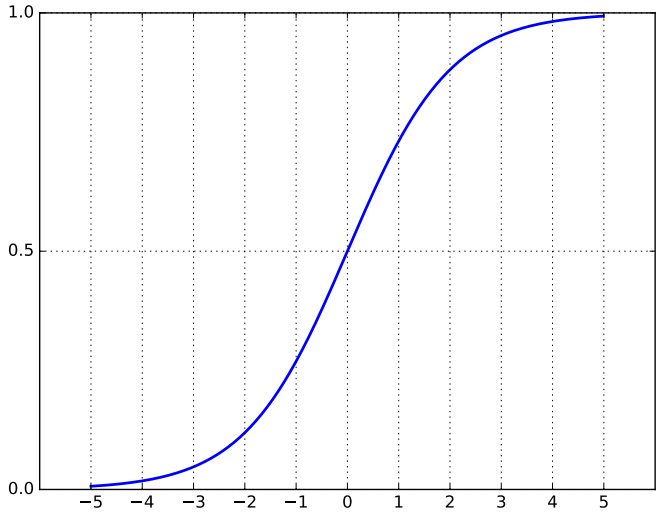


Figure: Sigmoid activation function σ

$$\sigma : x \mapsto \frac{\exp(x)}{1 + \exp(x)}$$

- Saturated function due to horizontal asymptotes:
  - Gradient is close to zero in these two areas ($\pm\infty$)
  - Rescaling the inputs of each layer can help to avoid these areas.

Sigmoid activation function

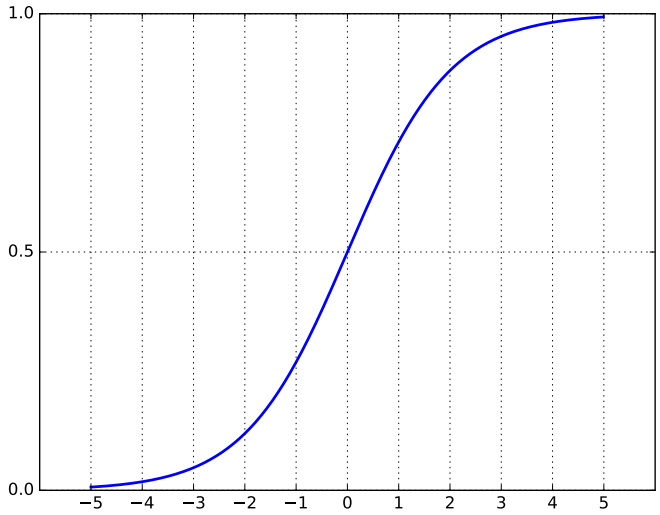


Figure: Sigmoid activation function σ

$$\sigma : x \mapsto \frac{\exp(x)}{1 + \exp(x)}$$

- Sigmoid is not a zero-centered function
  - Rescaling data
- Computing $\exp(x)$ is a bit costly

Hyperbolic tangent

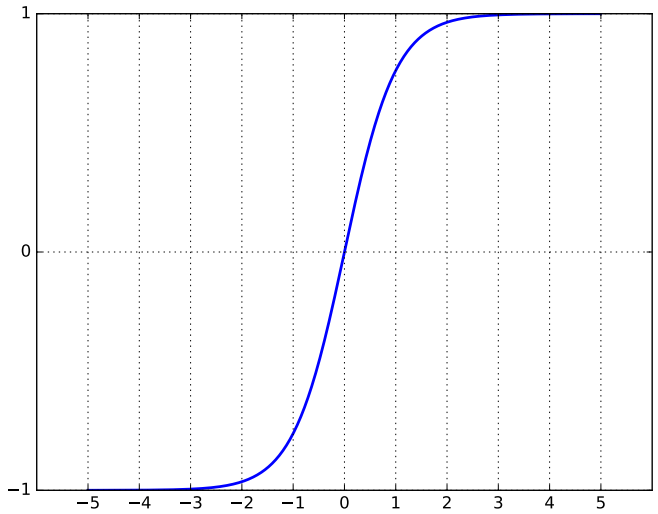


Figure: Hyperbolic tangent (tanh)

$$\tanh : x \mapsto \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- The function tanh is zero-centered
  - No need for rescaling data

Hyperbolic tangent

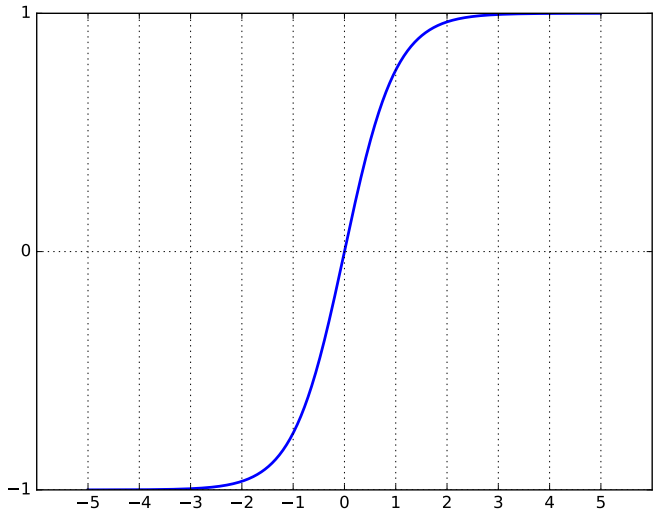


Figure: Hyperbolic tangent (tanh)

$$\tanh : x \mapsto \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- Saturated function due to horizontal asymptotes:
  - Gradient is close to zero in these two areas ($\pm\infty$)
  - Rescaling the inputs of each layer can help to avoid these areas.

Hyperbolic tangent

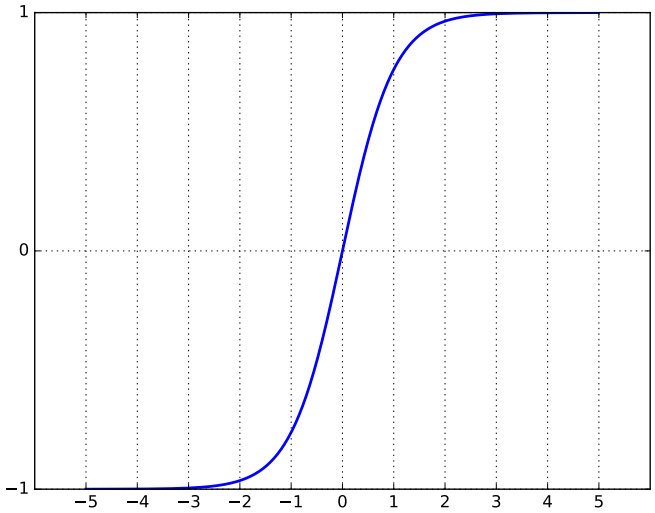


Figure: Hyperbolic tangent (tanh)

$$\tanh : x \mapsto \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- Computing $\exp(x)$ is a bit costly
- Note that $\tanh(x) = 2\sigma(2x) - 1$

Rectified Linear Unit (ReLU)

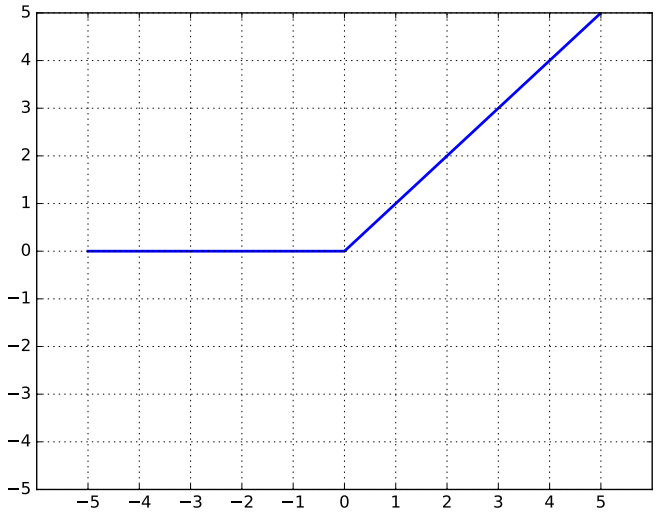


Figure: Rectified Linear Unit (ReLU)

$$ReLU : x \mapsto \max(0, x)$$

- Not a saturated function in $+\infty$
- But saturated/null in the region $x \leq 0$
- Computationally efficient
- Training NN with ReLU is faster than with sigmoid/tanh.
- Biologically plausible

More on ReLU

The idea of ReLU in neural networks seems to appear in ["Cognitron: A self-organizing multilayered neural network"; "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition", Fukushima 1975; Fukushima and Miyake 1982].

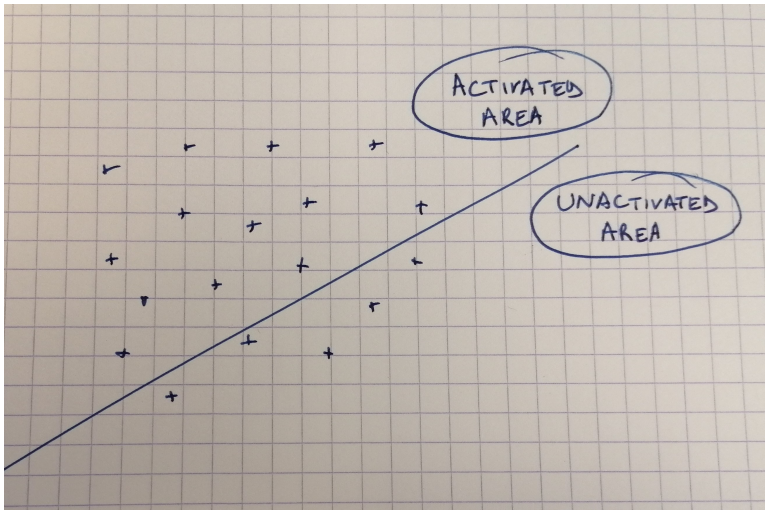


Figure: Good parameter initialization - ReLU is active

More on ReLU

The idea of ReLU in neural networks seems to appear in ["Cognitron: A self-organizing multilayered neural network"; "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition", Fukushima 1975; Fukushima and Miyake 1982].

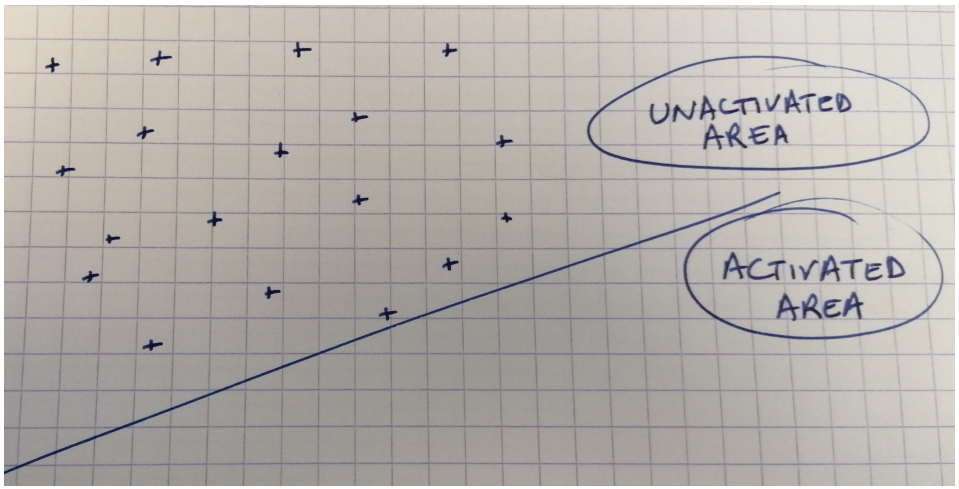


Figure: Bad parameter initialization - ReLU outputs zero

More on ReLU

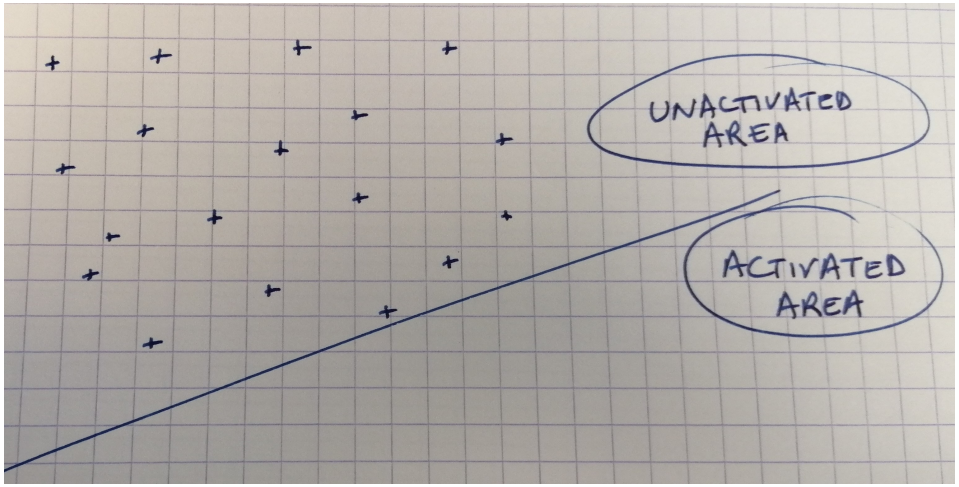


Figure: Bad parameter initialization - ReLU outputs zero

ReLU output can be zero but positive initial bias can help.

Related to biology ["Deep sparse rectifier neural networks", Glorot, Bordes, et al. 2011]:

- Most of the time, neurons are inactive.
- when they activate, their activation is proportional to their input.

Parametric ReLU

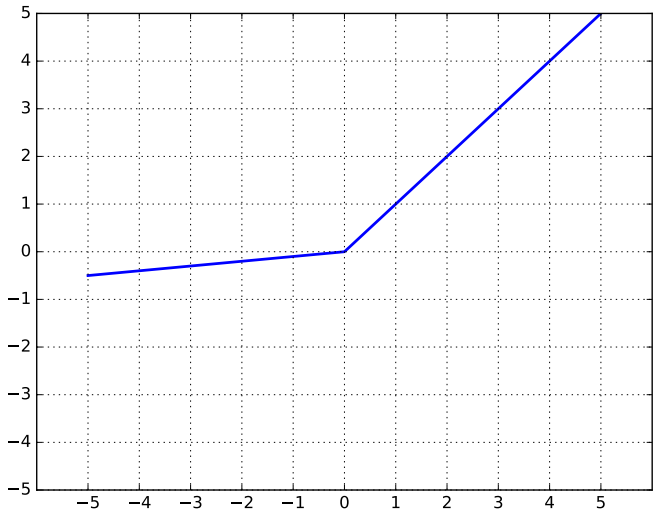


Figure: Parametric ReLU

$$\text{Parametric ReLU} : x \mapsto \max(\alpha x, x)$$

- Leaky ReLU: $\alpha = 0.1$

  ["Rectifier nonlinearities improve neural network acoustic models", Maas et al. 2013]

Parametric ReLU

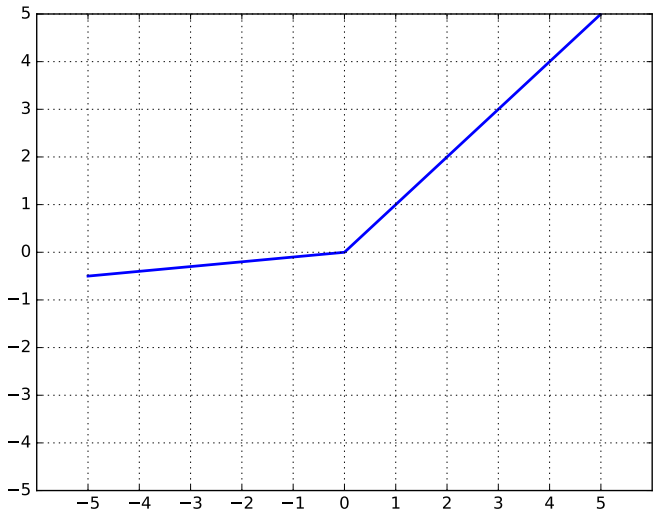


Figure: Parametric ReLU

$$\text{Parametric ReLU} : x \mapsto \max(\alpha x, x)$$

- Absolute Value Rectification:

$$\alpha = -1$$

["What is the best multi-stage architecture for object recognition?", Jarrett et al. 2009]

Parametric ReLU

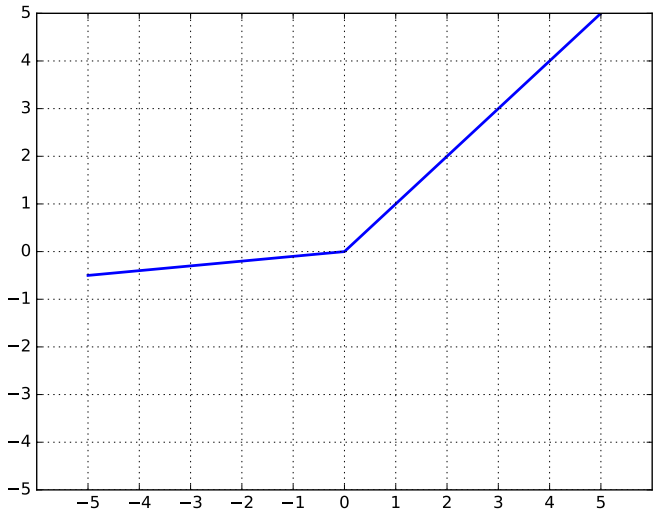


Figure: Parametric ReLU

$$\text{Parametric ReLU} : x \mapsto \max(\alpha x, x)$$

- Parametric ReLU: α optimized during backpropagation. Activation function is learned.

  ["Empirical evaluation of rectified activations in convolutional network", Xu et al. 2015]

Exponential Linear Unit (ELU)

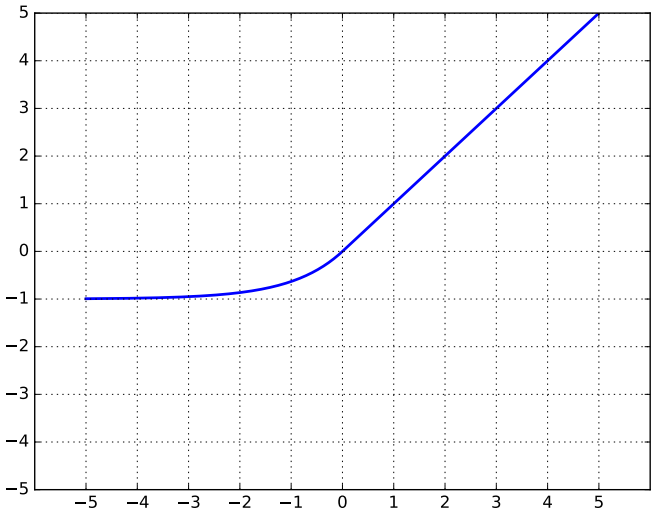


Figure: Exponential Linear Unit (ELU)

$$ELU : x \mapsto \begin{cases} x & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{otherwise} \end{cases}$$

- Close to ReLU but differentiable
- Closer to zero mean output.
- α is set to 1.0.
- Robustness to noise

  ["Fast and accurate deep network learning by exponential linear units (elus)", Clevert et al. 2015]

Maxout

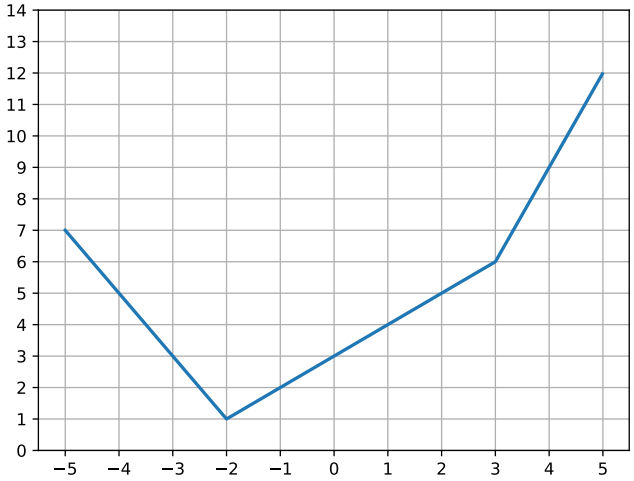


Figure: Maxout activation function, with $k = 3$ pieces

$$x \mapsto \max(w_1 x + b_1, w_2 x + b_2, w_3 x + b_3)$$

- Learn piecewise linear functions with k pieces: no saturation.

Maxout

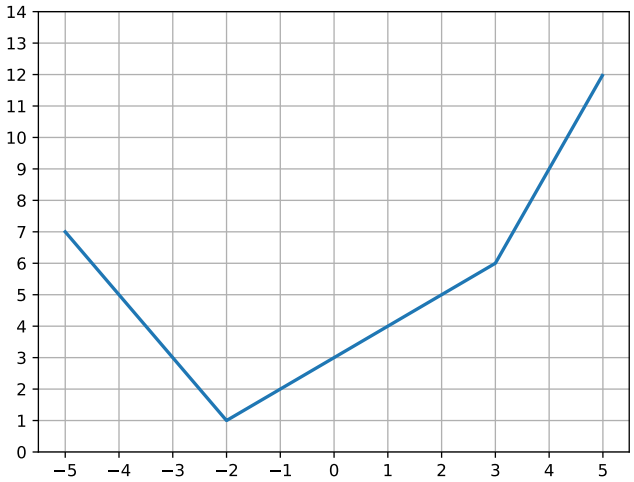


Figure: Maxout activation function, with $k = 3$ pieces

$$x \mapsto \max(w_1 x + b_1, w_2 x + b_2, w_3 x + b_3)$$

- Number of parameters multiplied by k

  ["Maxout networks", Goodfellow, Warde-Farley, et al. 2013] ["Deep maxout neural networks for speech recognition", Cai et al. 2013]

- Resist to catastrophic forgetting

  ["An empirical investigation of catastrophic forgetting in gradient-based neural networks", Goodfellow, Mirza, et al. 2013]

Swish

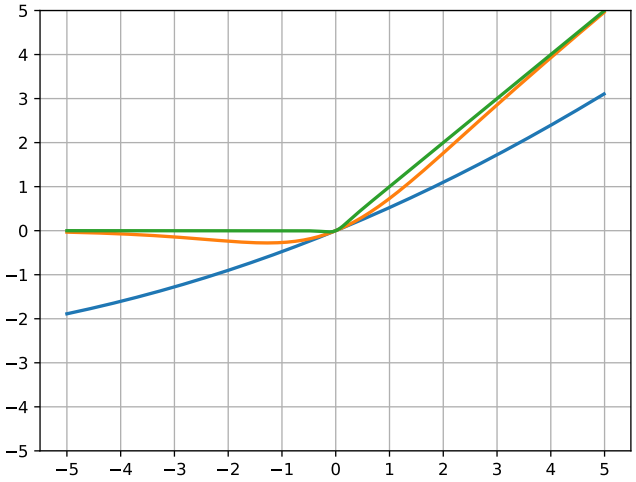


Figure: Swish function for $\beta = 0.1, 1, 10$

$$Swish : x \mapsto x \frac{\exp(\beta x)}{1 + \exp(\beta x)}$$

- Swish interpolates between the linear function and ReLU.
- ["Searching for activation functions", Ramachandran et al. 2017]
- Non-monotonic function - seems to be an important feature.

Conclusion on activation functions

- Use ReLU (or Swish).
- Test Leaky ReLU, maxout, ELU.
- Try out Tanh, but do not expect too much.
- Do not use sigmoid.

Outline

Output units

- Linear output unit:

$$\hat{y} = W^T h + b$$

$\rightarrow$ Linear regression based on the new variables h.

Output units

- Sigmoid output unit, used to predict $\{0, 1\}$ outputs:

$$\mathbb{P}(Y = 1|h) = \sigma(W^T h + b),$$

where $\sigma(t) = e^t/(1 + e^t)$.

$\rightarrow$ Logistic regression based on the new variables h.

- Softmax output unit, used to predict $\{1, \ldots, K\}$:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

where, each z_i is the activation of one neuron of the previous layer, given by $z_i = W_i^T h + b_i$.

$\rightarrow$ Multinomial logistic regression based on the new variables h.

Multinomial logistic regression

Generalization of logistic regression for multiclass outputs: for all $1 \leq k \leq K$,

$$\log\left(\frac{\mathbb{P}[Y_i = k]}{Z}\right) = \beta_k X_i, \tag{5}$$

Hence, for all $1 \leq k \leq K$,

$$\mathbb{P}[Y_i = k] = Z e^{\beta_k X_i}, \tag{6}$$

where

$$Z = \frac{1}{\sum_{k=1}^{K} e^{\beta_k X_i}}. \tag{7}$$

Thus,

$$\mathbb{P}[Y_i = k] = \frac{e^{\beta_k X_i}}{\sum_{\ell=1}^{K} e^{\beta_\ell X_i}}. \tag{8}$$

Biology bonus

Softmax, used with cross-entropy:

$$
\begin{aligned}
& -\log(\mathbb{P}(Y=y|z)) && (9)\\
=& -\log \operatorname{softmax}(z)_y && (10)\\
=& -z_y + \log\left(\sum_j e^{z_j}\right) && (11)\\
\simeq& \max_j z_j - z_y, && (12)
\end{aligned}
$$

Biology bonus

Softmax, used with cross-entropy:

$$
\begin{aligned}
& -\log(\mathbb{P}(Y = y|z)) && (9)\\
=& -\log \operatorname{softmax}(z)_y && (10)\\
=& -z_y + \log\left(\sum_j e^{z_j}\right) && (11)\\
\simeq& \max_j z_j - z_y, && (12)
\end{aligned}
$$

No contribution to the cost when $\operatorname{softmax}(z)_{\hat{y}}$ is maximal.

Biology bonus

Softmax, used with cross-entropy:

$$
\begin{aligned}
& -\log(\mathbb{P}(Y=y|z)) && (9)\\
=& -\log \operatorname{softmax}(z)_y && (10)\\
=& -z_y + \log\left(\sum_j e^{z_j}\right) && (11)\\
\simeq& \max_j z_j - z_y, && (12)
\end{aligned}
$$

No contribution to the cost when $\operatorname{softmax}(z)_{\hat{y}}$ is maximal.

Lateral inhibition: believed to exist between nearby neurons in the cortex. When the difference between the max and the other is large, winner takes all: one neuron is set to 1 and the others go to zero.

Biology bonus

Softmax, used with cross-entropy:

$$
\begin{aligned}
& -\log(\mathbb{P}(Y=y|z)) && (9)\\
=& -\log \operatorname{softmax}(z)_y && (10)\\
=& -z_y + \log\left(\sum_j e^{z_j}\right) && (11)\\
\simeq& \max_j z_j - z_y, && (12)
\end{aligned}
$$

No contribution to the cost when $\operatorname{softmax}(z)_{\hat{y}}$ is maximal.

More complex models: Conditional Gaussian Mixture: Y is multimodal ["On supervised learning from sequential data with applications for speech recognition"; "Generating sequences with recurrent neural networks", Schuster 1999; Graves 2013].

Outline

Cost functions

- Mean Square Error (MSE)

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i, f_\theta(\mathbf{X}_i)) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - f_\theta(\mathbf{X}_i))^2$$

- Mean Absolute Error

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i, f_\theta(\mathbf{X}_i)) = \frac{1}{n}\sum_{i=1}^{n}|Y_i - f_\theta(\mathbf{X}_i)|$$

- $0-1$ Error

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i, f_\theta(\mathbf{X}_i)) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{Y_i \neq f_\theta(\mathbf{X}_i)}$$

Cost functions

Cross entropy (or negative log-likelihood):

$$\ell(y_i, f_\theta(\mathbf{x}_i)) = -\log\left([f_\theta(\mathbf{x}_i)]_{y_i}\right) \qquad (13)$$

- Very popular!

Cost functions

Cross entropy (or negative log-likelihood):

$$\ell(y_i, f_\theta(\mathbf{x}_i)) = -\log\left([f_\theta(\mathbf{x}_i)]_{y_i}\right) \qquad (13)$$

- Should help to prevent saturation:

$$-\log(\mathbb{P}(Y = y_i|\mathbf{X} = \mathbf{x}_i)) \qquad (14)$$
$$= -\log(\sigma((2y-1)(W^T h + b))),$$

with

$$\sigma(t) = \frac{e^t}{1+e^t}$$

Usually, saturation occurs when $(2y-1)(W^T h + b) \ll -1$. In this case, $-\log(\mathbb{P}(Y = y_i|X))$ is linear in W and b, therefore preventing saturation to happen.

Cost functions

Cross entropy (or negative log-likelihood):

$$\ell(y_i, f_\theta(\mathbf{x}_i)) = -\log\left([f_\theta(\mathbf{x}_i)]_{y_i}\right) \qquad (13)$$

- Mean Square Error should not be used with softmax output units

  ["Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition", Bridle 1990]

Outline

Weight initialization

First idea: Set all weights and bias to the same value.

Small or big weights?

Consider the initial weight distribution to be $\mathcal{N}(0, \sigma^2)$.

1. If the variance of the weights is too small, that is $\sigma^2 \ll 1$:
   - the output of each neuron is close to a dirac in 0: there is no activation at all.
2. If the variance of the weights is too large, that is $\sigma^2 \gg 1$:
   - the linear combinations are very large, which increases the saturation phenomenon.
3. In any case, no need to tune the bias: they can be initially set to zero.

Other initialization

Idea: the variance of the input should be the same as the variance of the output.

Let w_j be any weight between layer j and layer $j+1$.

1. He et al. initialization

   ["Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", He et al. 2015]

   Initialize bias to zero and weights randomly using

$$w_j \sim \mathcal{N}\left(0, \frac{\sqrt{2}}{n_j}\right),$$

   where n_j is the size of layer j.

Other initialization

Idea: the variance of the input should be the same as the variance of the output.

Let w_j be any weight between layer j and layer $j+1$.

1. Xavier initialization

   ["Understanding the difficulty of training deep feedforward neural networks", Glorot and Bengio 2010]

   Initialize bias to zero and weights randomly using

   $$w_j \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right],$$

   where n_j is the size of layer j

   $\rightarrow$ Not theoretically valid for ReLU

Other initialization

Idea: the variance of the input should be the same as the variance of the output.

Let w_j be any weight between layer j and layer $j+1$.

1. Xavier initialization

   ["Understanding the difficulty of training deep feedforward neural networks", Glorot and Bengio 2010]

   Initialize bias to zero and weights randomly using

   $$w_j \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right],$$

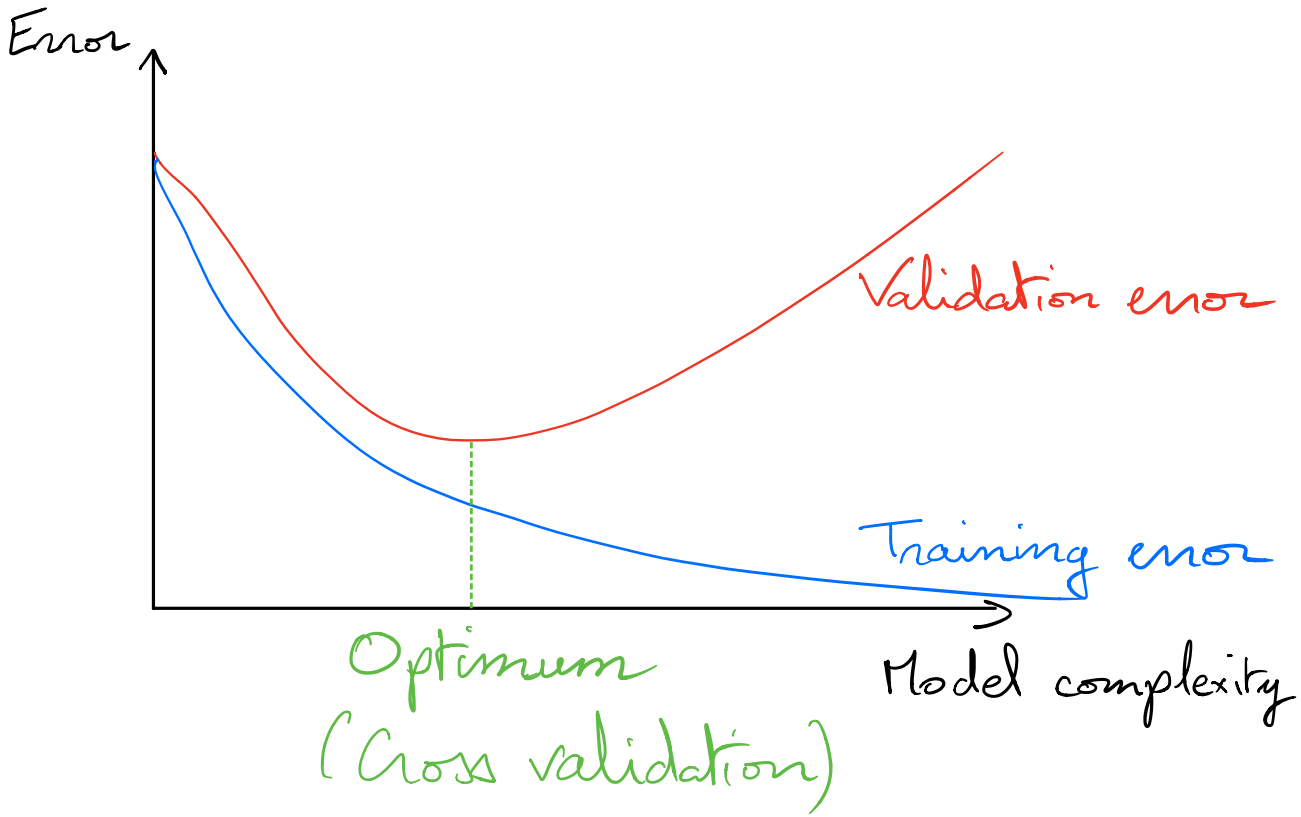
   where n_j is the size of layer j

   $\rightarrow$ Not theoretically valid for ReLU

Bonus: ["All you need is a good init", Mishkin and Matas 2015]

Outline

Regularizing to avoid overfitting



Avoid overfitting by imposing some constraints over the parameter space.

Reducing variance and increasing bias.

Avoiding overfitting

- Penalization (L1 or L2)
  Replacing the cost function $\mathcal{L}$ by $\tilde{\mathcal{L}}(\theta, X, y) = \mathcal{L}(\theta, X, y) + \text{pen}(\theta)$.
- Soft weight sharing - see CNN lecture
  Reduce the parameter space artificially by imposing explicit constraints.
- Dropout
  Randomly kill some neurons during optimization and predict with the full network.
- Batch normalization
  Renormalize a layer inside a batch, so that the network does not overfit on this particular batch.
- Early stopping
  Stop the gradient descent procedure when the error on the validation set increases.

Outline

Constraint the optimization problem

$$\min_{\theta} \mathcal{L}(\theta, X, y), \quad \text{s.t. } \text{pen}(\theta) \leq \textit{cste}. \quad (14)$$

Using Lagrangian formulation, this problem is equivalent to:

$$\min_{\theta} \mathcal{L}(\theta, X, y) + \lambda \, \text{pen}(\theta), \quad (15)$$

where

- $\mathcal{L}(\theta, X, y)$ is the loss function (data-driven term)
- pen is a function that increases when θ becomes more *complex* (penalty term)
- $\lambda \geq 0$ is a constant standing for the strength of the penalty term.

For Neural Networks, pen only penalizes the weights and not the bias: the latter being easier to estimate than weights.

Example of penalization

$$\min_{\theta} \mathcal{L}(\theta, X, y) + \text{pen}(\theta),$$

- Ridge

$$\text{pen}(\theta) = \lambda \|\theta\|_2^2$$

["Ridge regression: Biased estimation for nonorthogonal problems", Hoerl and Kennard 1970].

["Lecture notes on ridge regression", Wieringen 2015]

Example of penalization

$$\min_{\theta} \mathcal{L}(\theta, X, y) + \text{pen}(\theta),$$

- Lasso

$$\text{pen}(\theta) = \lambda \|\theta\|_1$$

["Regression shrinkage and selection via the lasso", Tibshirani 1996]

Example of penalization

$$\min_{\theta} \mathcal{L}(\theta, X, y) + \text{pen}(\theta),$$

- Elastic Net

$$\text{pen}(\theta) = \lambda \|\theta\|_2^2 + \mu \|\theta\|_1$$

["Regularization and variable selection via the elastic net", Zou and Hastie 2005]

Simple case: linear regression

Linear regression

The estimate of linear regression $\hat{\beta}$ is given by

$$\hat{\beta} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - \sum_{j=1}^{d} \beta_j x_i^{(j)})^2, \qquad (16)$$

which can be written as

$$\hat{\beta} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \| Y - \mathbb{X}\beta \|_2^2, \qquad (17)$$

where $\mathbb{X} \in M_{n,d}(\mathbb{R})$.

Solution:

$$\hat{\beta} = (\mathbb{X}'\mathbb{X})^{-1}\mathbb{X}'Y. \qquad (18)$$

Penalized regression

Penalized linear regression

The estimate of linear regression $\hat{\beta}_{\lambda,q}$ is given by

$$\hat{\beta}_{\lambda,q} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \|Y - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_q^q.$$

- $q = 2$: Ridge linear regression
- $q = 1$: LASSO

Ridge regression, $q = 2$

Ridge linear regression

The ridge estimate $\hat{\beta}_{\lambda,2}$ is given by

$$\hat{\beta}_{\lambda,2} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \|Y - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_2^2.$$

Solution:

$$\hat{\beta}_{\lambda,2} = (\mathbb{X}'\mathbb{X} + \lambda I)^{-1}\mathbb{X}'Y. \qquad (19)$$

This estimate has a bias equal to $-\lambda(\mathbb{X}'\mathbb{X} + \lambda I)^{-1}\beta$, and a variance $\sigma^2(\mathbb{X}'\mathbb{X} + \lambda I)^{-1}\mathbb{X}'\mathbb{X}(\mathbb{X}'\mathbb{X} + \lambda I)^{-1}$. Note that

$$\mathbb{V}[\hat{\beta}_{\lambda,2}] \leq \mathbb{V}[\hat{\beta}].$$

In the case of orthonormal design ($\mathbb{X}'\mathbb{X} = I$), we have

$$\hat{\beta}_{\lambda,2} = \frac{\hat{\beta}}{1+\lambda} = \frac{1}{1+\lambda}\mathbb{X}'Y. \qquad (20)$$

Sparsity

There is another desirable property on $\hat{\beta}$

If $\hat{\beta}_j = 0$, then feature j has no impact on the prediction:

$$\hat{y} = \text{sign}(x^\top \hat{\beta} + \hat{b})$$

If we have many features (d is large), it would be nice if $\hat{\beta}$ contained **zeros**, and many of them

- Means that only **few** features are statistically relevant.
- Means that only **few** features are useful to predict the label

Leads to a simpler model, with a "reduced" dimension

How do we enforce sparsity in β?

Sparsity

Tempting to solve

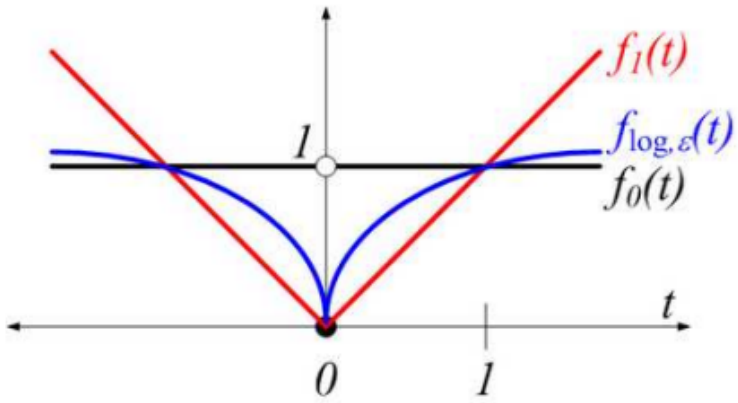
$$\hat{\beta}_{\lambda,0} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \| Y - \mathbb{X}\beta \|_2^2 + \lambda \|\beta\|_0. \quad (21)$$

where

$$\|\beta\|_0 = \#\{j \in \{1, \dots, d\} : \beta_j \neq 0\}.$$

To solve this, explore **all** possible supports of β. Too long! (NP-hard)

Find a convex proxy of $\|\|_0$: the ℓ_1-**norm** $\|\beta\|_1 = \sum_{j=1}^d |\beta_j|$



LASSO

Least Absolute Selection and Shrinkage Operator

Lasso linear regression

The LASSO estimate of linear regression $\hat{\beta}_{\lambda,1}$ is given by

$$\hat{\beta}_{\lambda,1} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \| Y - \mathbb{X}\beta \|_2^2 + \lambda \|\beta\|_1. \quad (22)$$

Solution: No close form in the general case

If the X_j are orthonormal then

$$\hat{\beta}_{\lambda,1,j} = X_j' Y \left(1 - \frac{\lambda}{2|X_j' Y|}\right)_+, \quad (23)$$

where $(x)_+ = \max(0, x)$.

Thus, in the very specific case of orthogonal design, we can easily show that L1 penalization implies a sparse vector if the parameter λ is properly tuned.

Outline

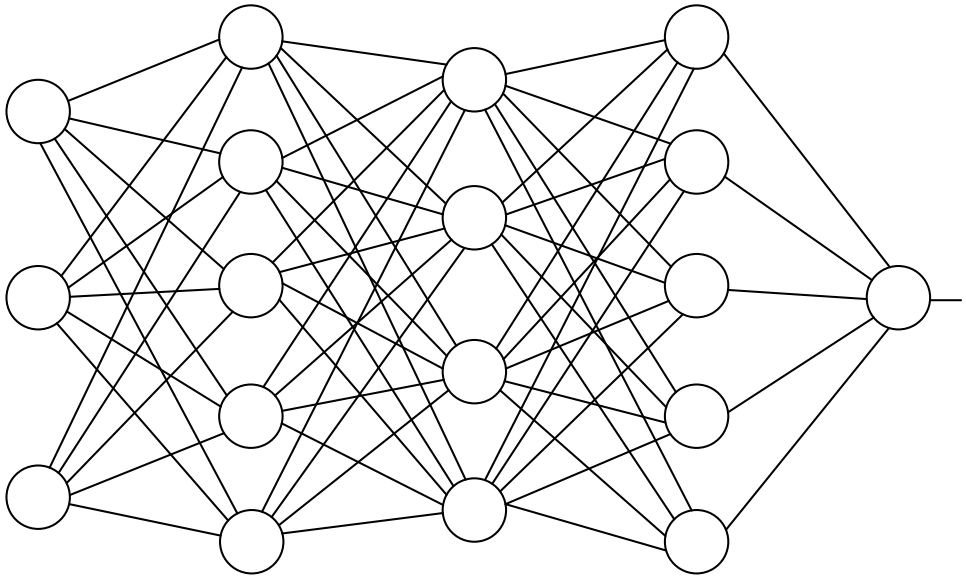
Dropout

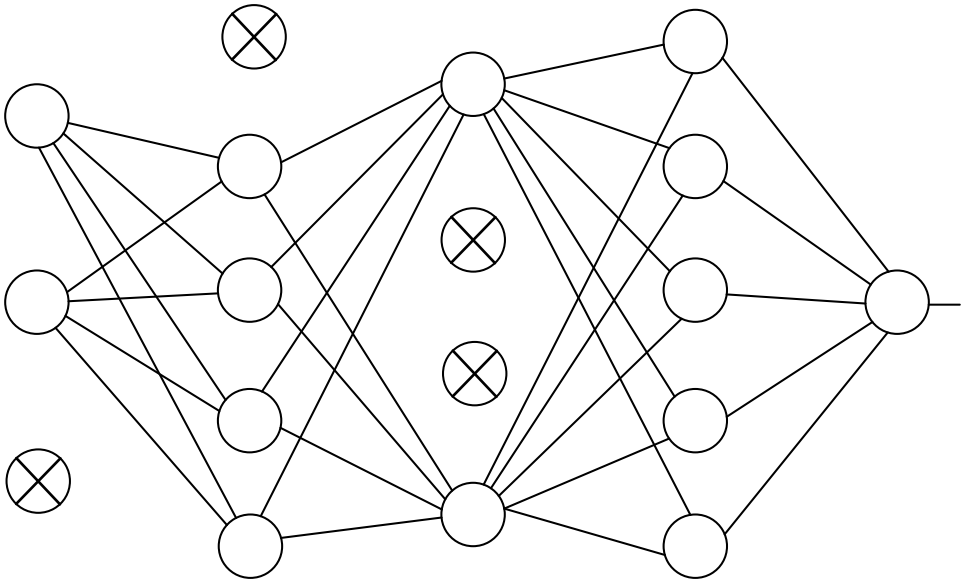
Dropout refers to dropping out units (hidden and visible) in a neural network, i.e., temporarily removing it from the network, along with all its incoming and outgoing connections.

Each unit is independently dropped with probability

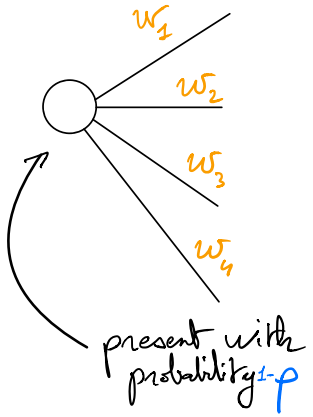
- $p = 0.5$ for hidden units
- $p \in [0, 0.5]$ for input units, usually $p = 0.2$.

Dropout

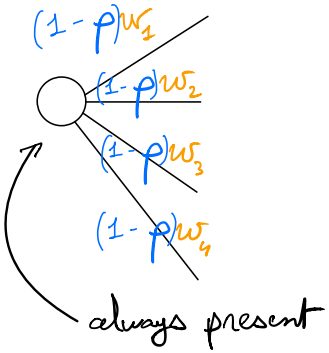




At train time



At test time



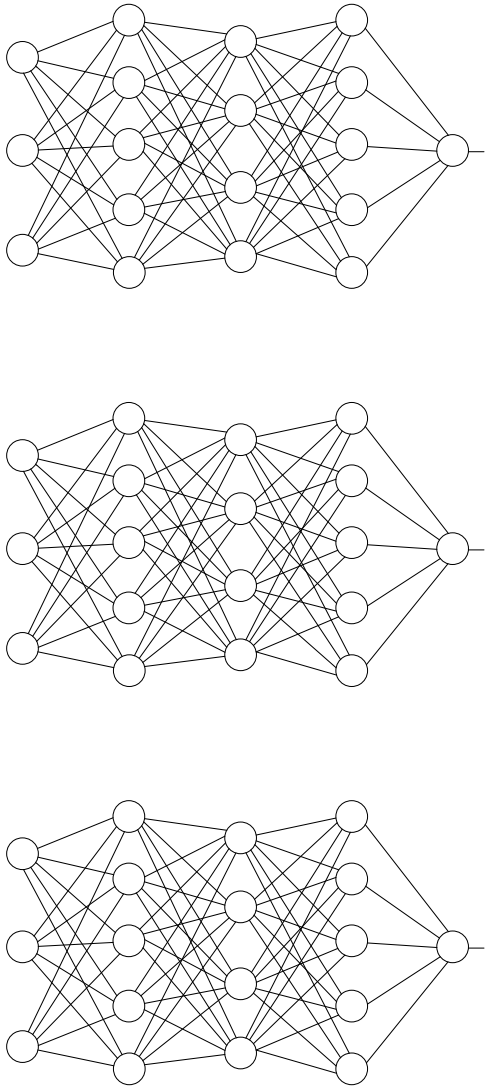
Dropout algorithm

Training step. While *not convergence*

1. Inside one epoch, for each mini-batch of size m,
   1. Sample m different mask. A mask consists in one Bernoulli per node of the network (inner and entry nodes but not output nodes). These Bernoulli variables are *i.i.d.*. Usually
      - ★ the probability of selecting an hidden node is 0.5
      - ★ the probability of selecting an input node is 0.8
   2. For each one of the m observation in the mini-batch,
      - ★ Do a forward pass on the masked network
      - ★ Compute backpropagation in the masked network
      - ★ Compute the average gradient
   3. Update the parameter according to the usual formula.

Prediction step.
Use all neurons in the network with weights given by the previous optimization procedure, times the probability p of being selected (0.5 for inner nodes, 0.8 for input nodes).

Another way of seeing dropout - Ensemble methods



Averaging many different neural networks.

Another way of seeing dropout - Ensemble methods

Averaging many different neural networks.
Different can mean either:

- randomizing the data set on which we train each network (via subsampling)

  Problem: not enough data to obtain good performance...

- building different network architectures and train each large network separately on the whole training set

  Problem: computationally prohibitive at training time and test time!

["Fast dropout training", Wang and Manning 2013]

["Dropout: A simple way to prevent neural networks from overfitting", Srivastava et al. 2014]

Dropping weights instead of the whole neurons: ["Regularization of neural networks using dropconnect", Wan et al. 2013]

Outline

Batch normalization

The network converges faster if its input are scaled (mean, variance) and decorrelated.

["Efficient backprop", LeCun et al. 1998]

Hard to decorrelate variables: requiring to compute covariance matrix.

["Batch normalization: Accelerating deep network training by reducing internal covariate shift", Ioffe and Szegedy 2015]

Ideas:

- Improving gradient flows
- Allowing higher learning rates
- Reducing strond dependence on initialization
- Related to regularization (maybe slightly reduces the need for Dropout)

Algorithm

See ["Batch normalization: Accelerating deep network training by reducing internal covariate shift", Ioffe and Szegedy 2015]

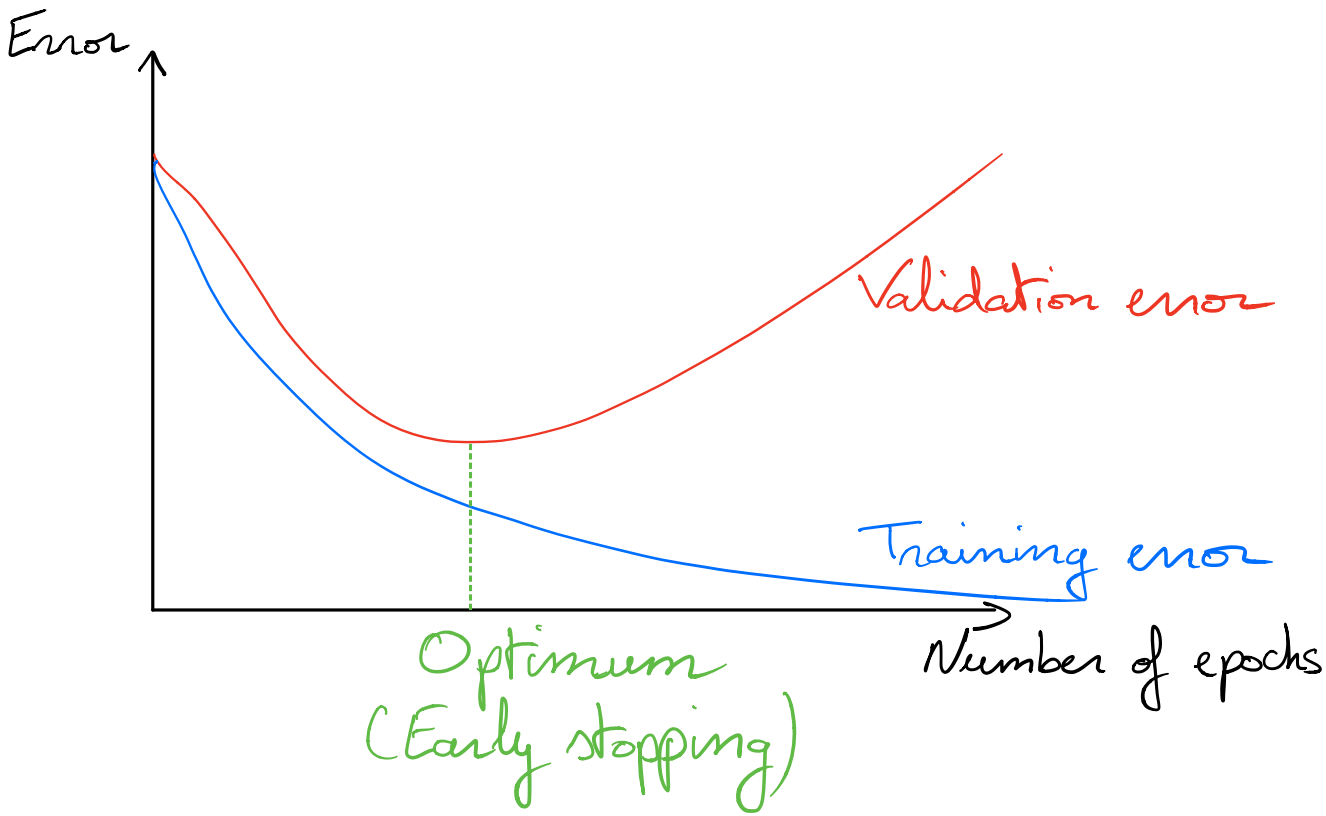
1. For every neuron k in the first layer, which outputs $x_i^{(k)}$ for the ith observation,
   1. $\mu_B^{(k)} = \frac{1}{m}\sum_{i=1}^m x_i^{(k)}$
   2. $\sigma_{B,k}^2 = \frac{1}{m}\sum_{i=1}^m (x_i^{(k)} - \mu_B^{(k)})^2$
   3. $\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_{B,k}^2 + \varepsilon}}$
   4. $y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)} \equiv BN_{\gamma^{(k)},\beta^{(k)}}(x_i^{(k)})$
2. $y_i^{(k)}$ is fed to the next layer and the procedure iterates.
3. Backpropagation is performed on the network parameters including $(\gamma^{(k)}, \beta^{(k)})$ for all $k = 1, \ldots, H_1$, where $H_1 \in \mathbb{N}$ is the number of neurons in the first layer.
4. For inference, compute the average over many training batches $\mathcal{B}$ of size m:
$$\mathbb{E}_{\mathcal{B}}[x^{(k)}] = \mathbb{E}_{\mathcal{B}}[\mu_{\mathcal{B}}^{(k)}] \quad \text{and} \quad \mathbb{V}_{\mathcal{B}}[x^{(k)}] = \frac{m}{m-1}\mathbb{E}_{\mathcal{B}}[\sigma_{\mathcal{B},k}^2].$$
5. For inference, replace every function $x^{(k)} \mapsto BN_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ in the network by
$$x^{(k)} \mapsto \gamma\left(\frac{x^{(k)} - \mathbb{E}_{\mathcal{B}}[x^{(k)}]}{\sqrt{\mathbb{V}_{\mathcal{B}}[x^{(k)}] + \varepsilon}}\right) + \beta^{(k)}.$$

Outline

Early stopping

Idea:

- Store the parameter values that lead to the lowest error on the validation set
- Return these values rather than the latest ones.



Early stopping algorithm

Parameters:

- patience p of the algorithm: number of times to observe no improvement on the validation set error before giving up;
- the number of steps n between evaluations.

Early stopping algorithm

1. Start with initial random values θ_0.
2. Let $\theta^\star = \theta_0$, $\mathrm{Err}^\star = \infty$, $j = 0$, $i = 0$.
3. While $j < p$
   1. Update θ by running the training algorithm for n steps
   2. $i = i + n$
   3. Compute the error $\mathrm{Err}(\theta)$ on the validation set
   4. If $\mathrm{Err}(\theta) < \mathrm{Err}^\star$
      - ★ $\theta^\star = \theta$
      - ★ $\mathrm{Err}^\star = \mathrm{Err}(\theta)$
      - ★ $j = 0$

      else $j = j + 1$.
4. Return $\theta^\star$ and the overall number of steps $i^\star = i - np$.

How to leverage on early stopping?

First idea: use early stopping to determine the best number of iterations $i^\star$ and train on the whole data set for $i^\star$ iterations.

Let $X^{(train)}, y^{(train)}$ be the training set.

- Split $X^{(train)}, y^{(train)}$ into $X^{(subtrain)}, y^{(subtrain)}$ and $X^{(valid)}, y^{(valid)}$.
- Run early stopping algorithm starting from random θ using $X^{(subtrain)}, y^{(subtrain)}$ for training data and $X^{(valid)}, y^{(valid)}$ for validation data. This returns $i^\star$ the optimal number of steps.
- Set θ to random values again.
- Train on $X^{(train)}, y^{(train)}$ for $i^\star$ steps.

How to leverage on early stopping?

Second idea: find the training error and the best parameters via early stopping. Starting from these parameters, train on the whole data set until the error matches the previous training error.

Let $X^{(train)}, y^{(train)}$ be the training set.

- Split $X^{(train)}, y^{(train)}$ into $X^{(subtrain)}, y^{(subtrain)}$ and $X^{(valid)}, y^{(valid)}$.
- Run early stopping algorithm starting from random θ using $X^{(subtrain)}, y^{(subtrain)}$ for training data and $X^{(valid)}, y^{(valid)}$ for validation data. This returns the optimal parameters $\theta^\star$.
- Set $\varepsilon = \mathcal{L}(\theta^\star, X^{(subtrain)}, y^{(subtrain)})$.
- While $\mathcal{L}(\theta^\star, X^{(valid)}, y^{(valid)}) > \varepsilon$, train on $X^{(train)}, y^{(train)}$ for n steps.

To go further

- Early stopping is a very old idea
  - ["Three topics in ill-posed problems", Wahba 1987]
  - ["A formal comparison of methods proposed for the numerical solution of first kind integral equations", Anderssen and Prenter 1981]
  - ["Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping", Caruana et al. 2001]
- But also an active area of research
  - ["Adaboost is consistent", Bartlett and Traskin 2007]
  - ["Boosting algorithms as gradient descent", Mason et al. 2000]
  - ["On early stopping in gradient descent learning", Yao et al. 2007]
  - ["Boosting with early stopping: Convergence and consistency", Zhang, Yu, et al. 2005]
  - ["Early stopping for kernel boosting algorithms: A general analysis with localized complexities", Wei et al. 2017]

More on reducing overfitting

- Soft-weight sharing:
  ["Simplifying neural networks by soft weight-sharing", Nowlan and Hinton 1992]
- Model averaging:
  Average over: random initialization, random selection of minibatches, hyperparameters, or outcomes of nondeterministic neural networks.
- Boosting neural networks by incrementally adding neural networks to the ensemble
  ["Training methods for adaptive boosting of neural networks", Schwenk and Bengio 1998]
- Boosting has also been applied interpreting an individual neural network as an ensemble, incrementally adding hidden units to the networks
  ["Convex neural networks", Bengio et al. 2006]

Outline

Pipeline for neural networks

- Step 1: Preprocess/normalize the data.
- Step 2: Choose the NN architecture (number of layers, number of nodes per layer...)
- Step 3: train the network
- Step 4: Find the best learning rate (LR)
  1. The error does not change too much (LR too small) or the error explodes, NaN (LR too high).
  2. Find a rough range $[10^{-5}, 10^{-3}]$.
- Sanity checks:
  1. Compare the NN loss to that of a dummy classifier.
  2. Increasing regularization should increase the training set error
  3. A NN trained on a small fraction of the data should overfit.

Playing with neural network:

http://playground.tensorflow.org/

[Aiz64] Mark A Aizerman. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and remote control* 25 (1964), pp. 821–837.

[AP81] RS Anderssen and PM Prenter. "A formal comparison of methods proposed for the numerical solution of first kind integral equations". In: *The ANZIAM Journal* 22.4 (1981), pp. 488–500.

[Ben+06] Yoshua Bengio et al. "Convex neural networks". In: *Advances in neural information processing systems*. 2006, pp. 123–130.

[Bla+20] Davis Blalock et al. "What is the state of neural network pruning?" In: *arXiv preprint arXiv:2003.03033* (2020).

[Blo62] Hans-Dieter Block. "The perceptron: A model for brain functioning. i". In: *Reviews of Modern Physics* 34.1 (1962), p. 123.

[Bri90] John S Bridle. "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition". In: *Neurocomputing*. Springer, 1990, pp. 227–236.

[BS85] Widrow Bernard and D Stearns Samuel. "Adaptive signal processing". In: *Englewood Cliffs, NJ, Prentice-Hall, Inc* 1 (1985), p. 491.

[BT07] Peter L Bartlett and Mikhail Traskin. "Adaboost is consistent". In: *Journal of Machine Learning Research* 8.Oct (2007), pp. 2347–2368.

[CLG01] Rich Caruana, Steve Lawrence, and C Lee Giles. "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping". In: *Advances in neural information processing systems*. 2001, pp. 402–408.

[CSL13] Meng Cai, Yongzhe Shi, and Jia Liu. "Deep maxout neural networks for speech recognition". In: *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE. 2013, pp. 291–296.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[Ege+20] Romain Egele et al. "AgEBO-Tabular: Joint Neural Architecture and Hyperparameter Search with Autotuned Data-Parallel Training for Tabular Data". In: *arXiv preprint arXiv:2010.16358* (2020).

[FM82] Kunihiko Fukushima and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[FS99] Yoav Freund and Robert E Schapire. "Large margin classification using the perceptron algorithm". In: *Machine learning* 37.3 (1999), pp. 277–296.

[Fuk75] Kunihiko Fukushima. "Cognitron: A self-organizing multilayered neural network". In: *Biological cybernetics* 20.3-4 (1975), pp. 121–136.

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.

[Goo+13a] Ian J Goodfellow, Mehdi Mirza, et al. "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *arXiv preprint arXiv:1312.6211* (2013).

[Goo+13b] Ian J Goodfellow, David Warde-Farley, et al. "Maxout networks". In: *arXiv preprint arXiv:1302.4389* (2013).

[Gra13] Alex Graves. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013).

[He+15] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[Heb49] DO Hebb. *The organization of behavior: a neuropsychological theory*. Wiley, 1949.

[Hin+12] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[HK70] Arthur E Hoerl and Robert W Kennard. "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 55–67.

[Hu64] Michael Jen-Chao Hu. "Application of the adaline system to weather forecasting". PhD thesis. Department of Electrical Engineering, Stanford University, 1964.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[JKL+09] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. "What is the best multi-stage architecture for object recognition?" In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 2146–2153.

[LeC+98] Yann LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[Mas+00] Llew Mason et al. "Boosting algorithms as gradient descent". In: *Advances in neural information processing systems*. 2000, pp. 512–518.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[MM15] Dmytro Mishkin and Jiri Matas. "All you need is a good init". In: *arXiv preprint arXiv:1511.06422* (2015).

[MP43] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[MP69] Marvin Minsky and Seymour Papert. "Perceptrons.". In: (1969).

[MR13] Mehryar Mohri and Afshin Rostamizadeh. "Perceptron mistake bounds". In: *arXiv preprint arXiv:1305.0208* (2013).

[NH92] Steven J Nowlan and Geoffrey E Hinton. "Simplifying neural networks by soft weight-sharing". In: *Neural computation* 4.4 (1992), pp. 473–493.

[Nov63] Albert B Novikoff. *On convergence proofs for perceptrons*. Tech. rep. STANFORD RESEARCH INST MENLO PARK CA, 1963.

[Ola96] Mikel Olazaran. "A sociological study of the official history of the perceptrons controversy". In: *Social Studies of Science* 26.3 (1996), pp. 611–659.

[Ros60] Frank Rosenblatt. "Perceptron simulation experiments". In: *Proceedings of the IRE* 48.3 (1960), pp. 301–309.

[Ros61] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[SB98] Holger Schwenk and Yoshua Bengio. "Training methods for adaptive boosting of neural networks". In: *Advances in neural information processing systems*. 1998, pp. 647–653.

[Sch99] Michael Schuster. "On supervised learning from sequential data with applications for speech recognition". In: *Daktaro disertacija, Nara Institute of Science and Technology* 45 (1999).

[Sri+14] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[TGK63] LR Talbert, GF Groner, and JS Koford. "Real-Time Adaptive Speech-Recognition System". In: *The Journal of the Acoustical Society of America* 35.5 (1963), pp. 807–807.

[Tib96] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288.

[Wah87] Grace Wahba. "Three topics in ill-posed problems". In: *Inverse and ill-posed problems*. Elsevier, 1987, pp. 37–51.

[Wan+13] Li Wan et al. "Regularization of neural networks using dropconnect". In: *International conference on machine learning*. 2013, pp. 1058–1066.

[WH60] Bernard Widrow and Marcian E Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.

[Wie15] Wessel N van Wieringen. "Lecture notes on ridge regression". In: *arXiv preprint arXiv:1509.09169* (2015).

[WM13] Sida Wang and Christopher Manning. "Fast dropout training". In: *international conference on machine learning*. 2013, pp. 118–126.

[WW88] Capt Rodney Winter and B Widrow. "Madaline Rule II: a training algorithm for neural networks". In: *Second Annual International Conference on Neural Networks*. 1988, pp. 1–401.

[WYW17] Yuting Wei, Fanny Yang, and Martin J Wainwright. "Early stopping for kernel boosting algorithms: A general analysis with localized complexities". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6067–6077.

[Xu+15] Bing Xu et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[YRC07] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. "On early stopping in gradient descent learning". In: *Constructive Approximation* 26.2 (2007), pp. 289–315.

[ZH05] Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320.

[ZY+05] Tong Zhang, Bin Yu, et al. "Boosting with early stopping: Convergence and consistency". In: *The Annals of Statistics* 33.4 (2005), pp. 1538–1579.