Grupo 8

Diego Bruno - Tomás Ferrer - Matías Heimann - Marcos Lund

Objetivo principal del trabajo

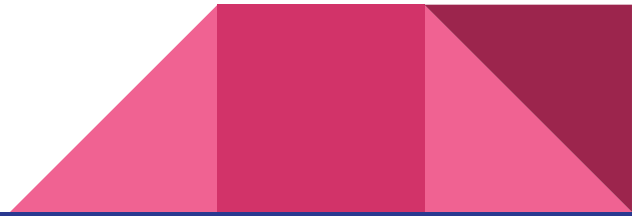
- Se propuso diseñar una red neuronal multicapa que aproxime un terreno dado en base a coordenadas del formato (latitud, longitud, altitud) que lo describen.
- Analizar los cambios que se realizan cambiando los parámetros e implementando mejoras.

Arquitectura utilizada

- Consideraciones:
  - Análisis de la cantidad de capas ocultas
  - Análisis de la cantidad de neuronas
  - Análisis de las salidas al cabo de cada época
- Incrementos de a cinco neuronas.
- Resultado: 10 neuronas en dos capas ocultas.

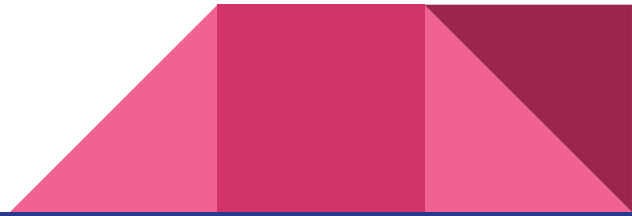
Implementación incremental

- Se eligen al azar un orden para los distintos patrones a utilizar.
- Se corrigen los pesos luego de cada patrón presentándose en un orden aleatorio.
- Una vez que la diferencia entre las salidas calculadas y las salidas obtenidas es menor a un epsilon se termina el entrenamiento.
- Puede aumentar el error entre épocas pero este tiende a disminuir.



Implementación batch

- Se eligen los patrones de la misma forma que en la incremental.
- Se corrigen los pesos de la red teniendo en cuenta los errores generados por todos los patrones de la red.
- Muy baja frecuencia de aumento de error de época a época.



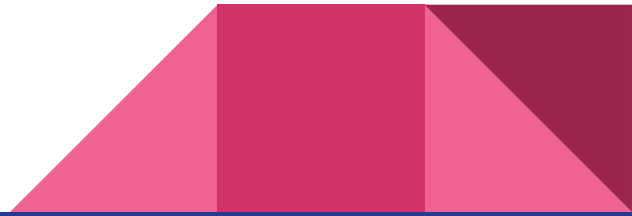
Funciones utilizadas y pesos iniciales

- La inicialización de los pesos de cada capa fueron elegidas con valores random del orden de $1/\sqrt{k_i}$ donde k_i hace referencia al número de entradas de la capa inferior a la actual.
- Funciones utilizadas
  - Tangente hiperbólica
  - Exponencial
- Normalización del dataset

Mejoras

Tanto para el método de entrenamiento incremental como en el batch se implementaron estas mejoras.

- Eta adaptativo
- Momentum
- ADAM (Adaptive Moment Estimation)



Momentum y Adam (implementación)

```
for i = 1:length(w)
  g{i} = (d{i + 1} * v{i}');
  switch (weight_optimization)
    # Default
    case 0
      dw{i} = eta * g{i};
    # Momentum
    case 1
      dw{i} = eta * g{i} + momentum_alpha * last_dw{i};
    # Adam
    case 2
      adam_m{i} = adam_beta1 * adam_m{i} + (1 - adam_beta1) * g{i};
      adam_v{i} = adam_beta2 * adam_v{i} + (1 - adam_beta2) * (g{i} .^ 2);
      m_hat = adam_m{i} / (1 - (adam_beta1 ^ epochs));
      v_hat = adam_v{i} / (1 - (adam_beta2 ^ epochs));
      dw{i} = eta * m_hat ./ (sqrt(v_hat) + adam_epsilon);
  endswitch
  w{i} = w{i} + dw{i};
endfor
last_dw = dw;
```

Resultados batch sin mejoras

- Peores resultados que en las otras configuraciones.
- Se utilizó con la función exponencial.
- Avanza muy lentamente.
- Con learning rates grandes no converge.
- Se necesita una gran cantidad de épocas para tener buenos resultados.
- Más patrones, menor error cuadrático medio final.

Resultados batch con mejoras

- Momentum
  - Se mostraron mejoras notables cuando el alpha se elige como 0.9
  - Con alphas menores a 0.9 tarda mucho más tiempo en alcanzar un error cuadrático medio aceptable
  - Con alphas mayores a 0.9 los resultados empiezan a empeorar a medida que este crece.
- Eta adaptativo
  - Mejora notable en comparación al batch sin mejoras.
  - Mejoría en comparación a Momentum
- ADAM
  - La mejora más útil.
  - Reduce ampliamente el nivel de generalización en comparación al ETA adaptativo y al Momentum.
  - Con beta1 = 0.9 y beta2 = 0.999 da los mejores resultados.

Resultados de Incremental sin mejora

- Se utilizó la función hiperbólica para tener un menor error.
- Con learning rates grandes no converge.
- Ante una mayor cantidad de patrones la generalización es más grande y el error más chico.
- Ante una mayor cantidad de épocas el error es más chico pero se puede dar una menor generalización debido a un sobre entrenamiento.

Resultados de Incremental con mejora

- Momentum
  - Se encontró una mejoría con valores como 0.9.
  - Los valores cercanos a 0.5 no eran útiles.
- Eta adaptativo
  - Mejoró con respecto al sin mejoras.
  - Se utilizaron valores de a y b pequeños.
- Momentum + Eta adaptativo
  - Con los valores pequeños de a y b y los valores de alpha recomendados en el momentum dieron los mejores resultados.
- ADAM
  - Dieron los mejores resultados.
  - Con beta1 = 0.9 y beta2 = 0.999 da los mejores resultados.

Comparación de funciones

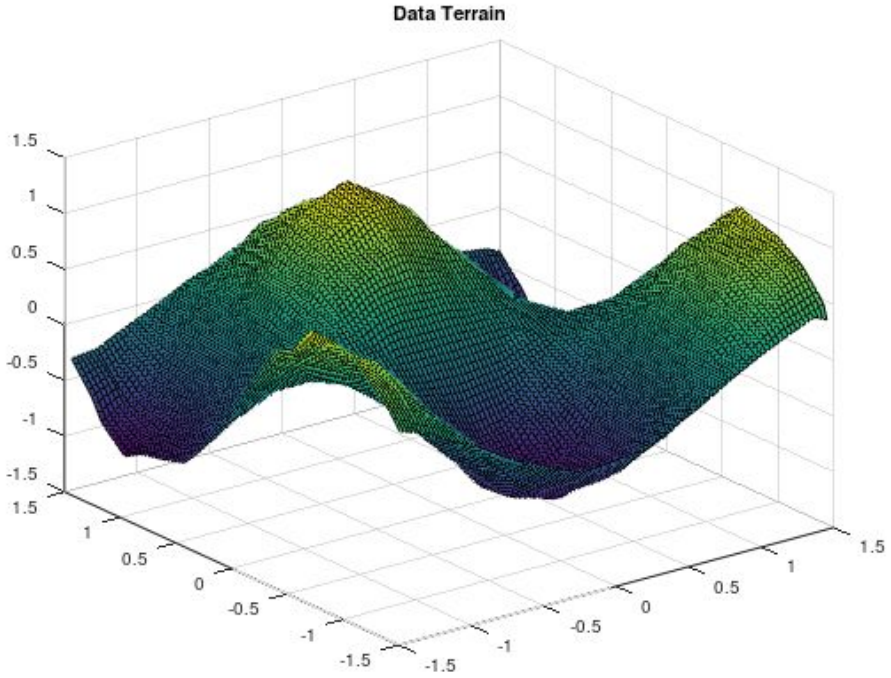
	Promedio de épocas	
Número de capas ocultas	**Función exponencial**	**Función tangente hiperbólica**
1	2186	> 40k
2	756	1231
3	1543	1704

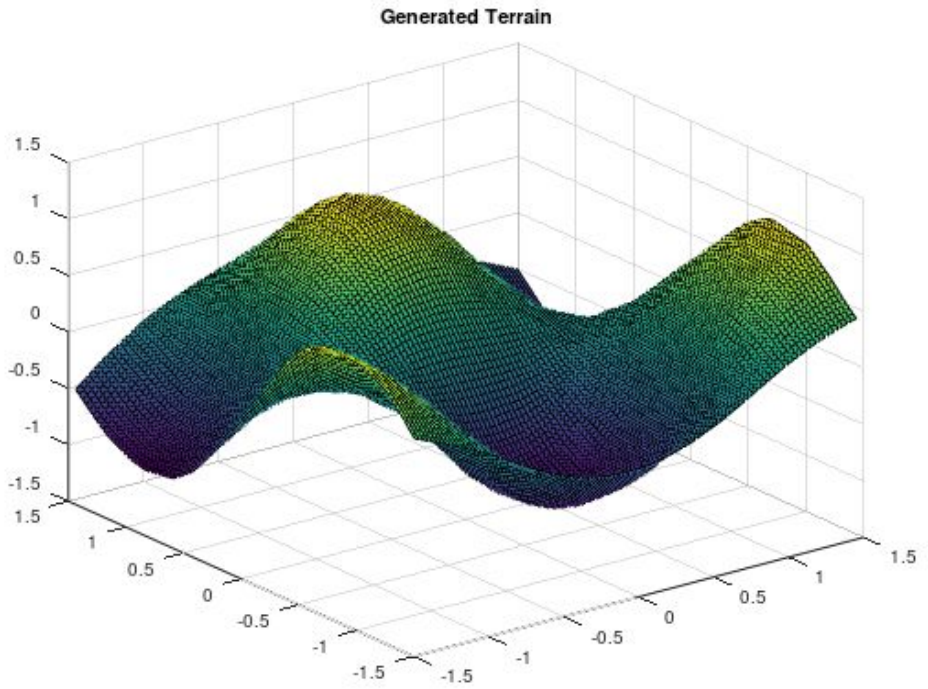
Resultados de batch con ADAM en función de las épocas

% Generalización (utilizando 0.05 de epsilon para considerar correctitud de salidas)		**Épocas**		
		500	**2000**	**5000**
Porcentaje de datos usados para entrenamiento	**0.1**	34.92%	40.14%	45.45%
	0.25	51.92%	71.65%	77.10%
	0.4	57.14%	74.38%	87.07%

Parámetros: Batch training. ETA = 0.01 con Adam y función exponencial, 2 capas de 12.

Generalización: 87.07% (params. tabla anterior)



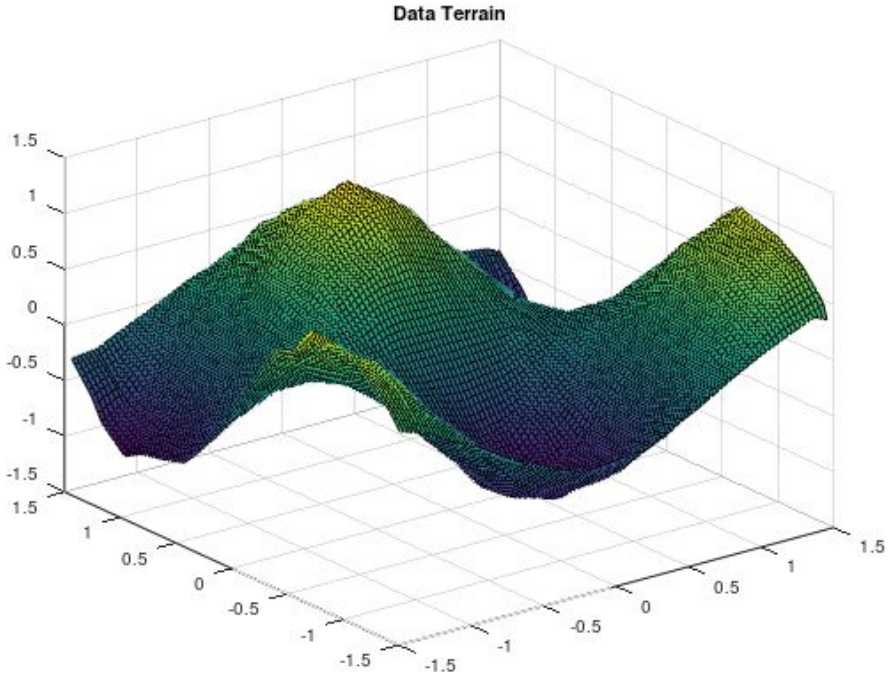


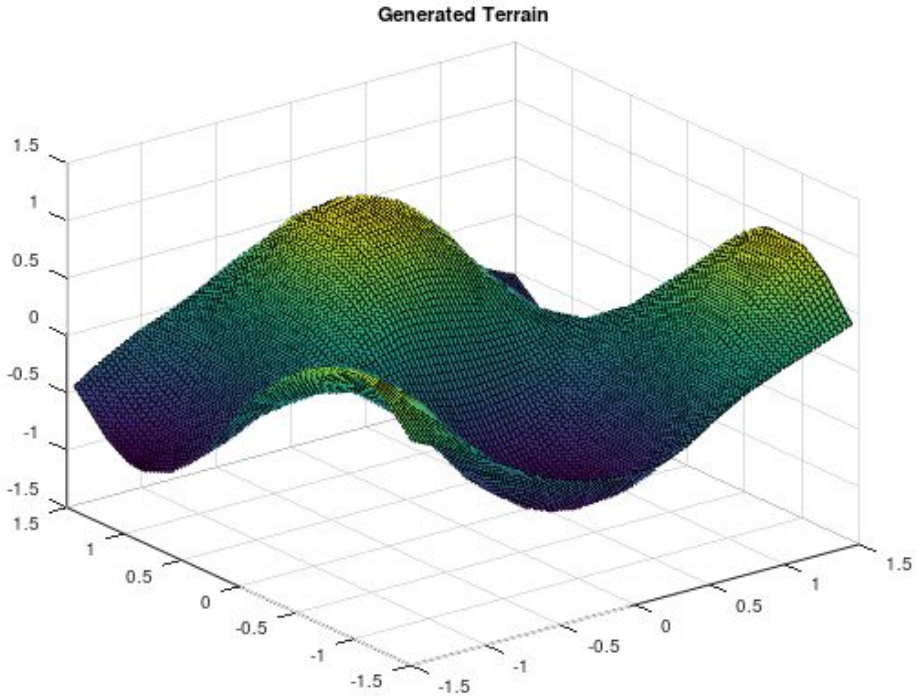
Resultados de Batch vs resultados de Incremental

% Generalización (utilizando 0.05 de epsilon para considerar correctitud de salidas)		Épocas	
		500 (batch)	500 (incremental)
Porcentaje de datos usados para entrenamiento	0.1	34.92%	41.72%
	0.25	51.92%	64.62%
	0.4	57.14%	78.01%

Parámetros: Batch training y Incremental training.
ETA = 0.01 con Adam y función exponencial, 2 capas de 12.

Generalización: 78.01% (params. tabla anterior)





Conclusiones

- Dos capas ocultas con diez neuronas por capa dieron los resultados más óptimos.
- La aplicación de mejoras al algoritmo de aprendizaje resultó positivo.
  - Para momentum un valor de alfa de 0.9 es el valor óptimo.
  - El eta adaptativo resultó más apto para la reducción de errores que el momentum.
  - ADAM resultó ser la mejor implementación con los valores de beta1 y beta2 recomendado.
- Un mayor uso de datos de entrenamiento conlleva a una mayor generalización, sin embargo, derivará en un tiempo requerido superior al deseado.